UCID- 20624

SIGNAL PROCESSING EXPERT CODE

by

Henry S. Ames

December 1985

Lawrence Livermore National Laboratory

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Price Code	Page Range
A01	Microfiche
Papercopy Prices	
A02	001-050
A03	051-100
A04	101-200
A05	201-300
A06	301-400
A07	401-500
A08	501-600
A09	601

TABLE OF CONTENTS

ABSTRACT

The purpose of this paper is to describe a prototype expert system called SPEC which was developed to demonstrate the utility of providing an intelligent interface for users of SIG, a general purpose signal processing code. The expert system is written in NIL, runs on a VAX 11/750 and consists of a backward chaining inference engine and an English-like parser. The inference engine uses knowledge encoded as rules about the formats of SIG commands and about how to perform frequency analyses using SIG. The system demonstrated that expert system can be used to control existing codes.

1.0 Introduction

In engineering there is a constant need to use signal processing techniques to analyze data, and there are a variety of general purpose codes available which can be used. Although a code may be very easy to use each person who wants to use it must learn the specific peculiarities of the code, perhaps in addition to learning how to do signal processing.

The technology of expert systems suggests that expert systems can be developed which understand how to run existing signal processing codes thereby freeing the user to concentrate his efforts on solving the problem instead of remembering the syntax of commands for specific signal processing codes. Such expert systems could also be given knowledge about signal processing so that the systems could give less experienced engineers guidance on signal processing issues.

It was decided that a prototype expert system, which finally became known as the Signal Processing Expert Code (SPEC), would be implemented to evaluate the utility of expert systems as a form of man-machine interface. SIG was chosen as the target signal processing code because it contains a general assortment of signal processing techniques and display capabilities, and has both a simple menu interface and a command language which can be driven from a file of commands. Potential users of SPEC were interested mostly in frequency domain analyses so the scope of the SPEC prototype was limited to those SIG commands which are appropriate for frequency analyses, such as Fourier transform, power spectral estimation and filtering commands.

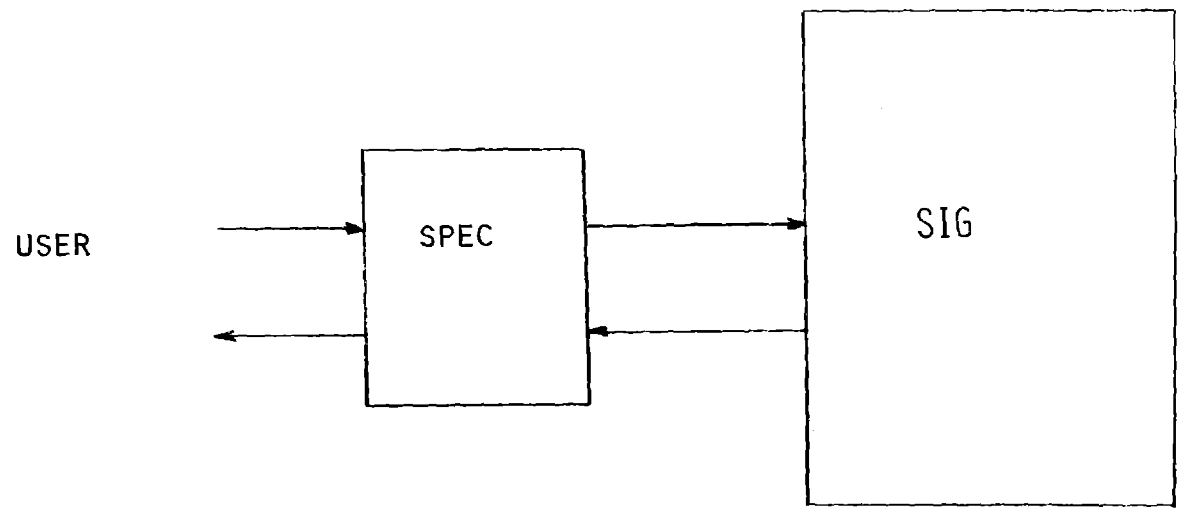
2.0 Overview of SPEC

The concept of the signal processing expert code (SPEC) is shown pictorially in figure 1. The user enters commands in an English-like syntax to the expert system which determines a goal, generates a plan to carry out the action, translates the plan into specific SIG commands and sends the commands to SIG. The entire system runs on a VAX under the VMS operating system. SPEC is written in a LISP from MIT, namely NIL, and SIG is written in FORTRAN. To utilize the graphics capability of SIG, the user must use a graphics terminal which is supported by SIG, such as a VT100 with Retro-graphics hardware.

The final version of SPEC consisted of about 2000 lines of LISP code, contained 34 rules and took several minutes to complete a request to plot an fft. Since it was only a prototype some features were not implemented; notably, the prototype was not able to analyze responses from SIG and did not provide the means to store or retrieve information about data files from session to session.

One of the objectives of this project was to NOT modify in any way the operations of the target signal processing code, namely SIG. This objective was attainable because SIG can read commands from a disk file. SPEC starts and stops SIG each time commands are transferred. In subsequent versions of such systems it is hoped that the expert

Figure 1. The concept of the Signal Processing Expert Code (SPEC)



system will be able to communicate with the target signal processing code using a pipe-like communications scheme so that the performance can be improved.

The protoype was developed initially to support users involved with the frequency analysis of environmental data, such as thermal couples and accelerometers. The functions which were included in the prototype are summarized in Table 1. The chosen functions are only a small part of the repertoire of SIG but were considered representative of the capabilities that potential users might need. With these functions the user can request the Fourier transform of a file of data, and the system will read the file, do the Fourier transform and plot the result.

3.0 Sample Session

In this section a sample session using SPEC is described in order to give the reader a good understanding of the capabilities of SPEC. The details of the dialogue are presented in Appendix A.

In the sample session, it is assumed that the user has two files of data in his disk directory, ABC and XYZ. These files contain voltage readings taken at equally spaced intervals of time. Furthermore, the files already contain a header at the beginning of each file which describes the starting time, time interval and format of the data. Since the file contains the header SPEC, will not ask the user for the information.

When SPEC is started, a welcome message and prompt are displayed. If the user wants to see the commands available, he need only type HELP. The user responses are underlined.

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
        LINE CONTINUATION IS &

?
```

The user decides to do a fast Fourier transform (fft) of the data in file ABC and to plot the result. The system then leads the user through several assumptions which the user approves by entering Y.

```
?   DO FFT ON FILE ABC <CR>

I WILL ASSUME THE FILE ABC CONTAINS TIME DATA ? (OK? Y OR N) Y <CR>

I WILL ASSUME THE FILE ABC IS ALREADY IN THE PROPER SIG FORMAT ?
(OK? Y OR N) Y <CR>
```

Table 1. Signal Processing Functions of SPEC

Read a file containing time domain data

Read a file containing frequency domain data

Fourier transform

Plot of time domain data

Plot of frequency domain data

Filter data using Bessel filter

Filter data using Butterworth filter

Inverse Fourier transform

Power spectral estimation using Welch's method

At this point SPEC generates the commands to cause SIG to read the data file. Next the system must find out the information necessary to do the fft.

```
I WILL USE ALL THE POINTS IN SIGNAL IN THE FFT ? (OK? Y OR N) Y <CR>
```

At this point SPEC generates the SIG commands to cause SIG to do an fft on the data previously read. Then the systems asks a question about the plot display, namely how to label the plot. However, in the prototype version the SIG interface was never modified to actually label the plots. The question is merely to indicate the capability of SPEC.

```
I WILL LABEL THE PLOT TEST ? (OK? Y OR N) Y <CR>
```

Finally, the system generates the plot and waits for the user to finish reviewing the data. When the user is finished he can hit a carriage return to cause SPEC to continue. The system plots the data as shown in Figure A.1.

Now, the user decides to see the original time domain data in the file ABC.

```
 ?  LET ME SEE THE TIME PLOT <CR>
```

The system asks the same question about labeling the plot and then displays the time data, as shown in Figure A.2.

Next, the user decides to plot the fft of his other file XYZ. So he first attempts to type a short version of the command.

```
?  NOW DO THE FILE XYZ <CR>
```

Since this sentence does not contain a keyword and the previous command was PLOT, the system assumes that the user wants to a plot of the file. The system asks for verification of the assumption.

```
I DON'T RECOGNIZE ANY WORD AS A COMMAND.
DO YOU MEAN TO DO THE COMMAND PLOT (Y OR N)?  N <CR>
```

But the user really wants to do an FFT and PLOT it.

```
WHAT DO YOU WANT TO DO?  FFT <CR>
```

The system then asks the same questions that were asked for the file ABC and eventually displays a plot of the fft as shown in Figure A.3.

Next the user decides to filter the data in the last file and to display the fft of the filtered data.

```
? FILTER FILE XYZ AND DO FFT AGAIN <CR>
```

Since the user did not specify what type of filter to use, the system attempts to find out if the user wants to use the default filter type, Bessel. Since the user does not know what types of filters are available he responds with N and asks for HELP.

```
I WILL USE A BESSEL FILTER ON THE FILE XYZ ? (OK? Y OR N)  N <CR>

WHAT TYPE OF FILTER DO YOU WANT TO USE ON FILE XYZ? HELP <CR>

VALID RESPONSES ARE AS FOLLOWS:
DEFAULT IS BESSEL
BESSEL BESS BES
BUTTERWORTH BUTTER BUT
HIT  <CR> TO CONTINUE <CR>

WHAT TYPE OF FILTER DO YOU WANT TO USE ON FILE XYZ? BESS <CR>
```

The system continues to ask the user questions regarding the characteristics of the filter and the size of the fft and eventually displays a plot of the fft of the filtered data as shown in Figure A.4.

The user continues with more dialogue which is described in Appendix A and finally decides to quit. Since the system was only intended as a prototype, software was not written to save any of the information about the files so the next session will not be aware of the transactions which have occurred.

```
?  QUIT <CR>

   ALL DONE

   NIL
```

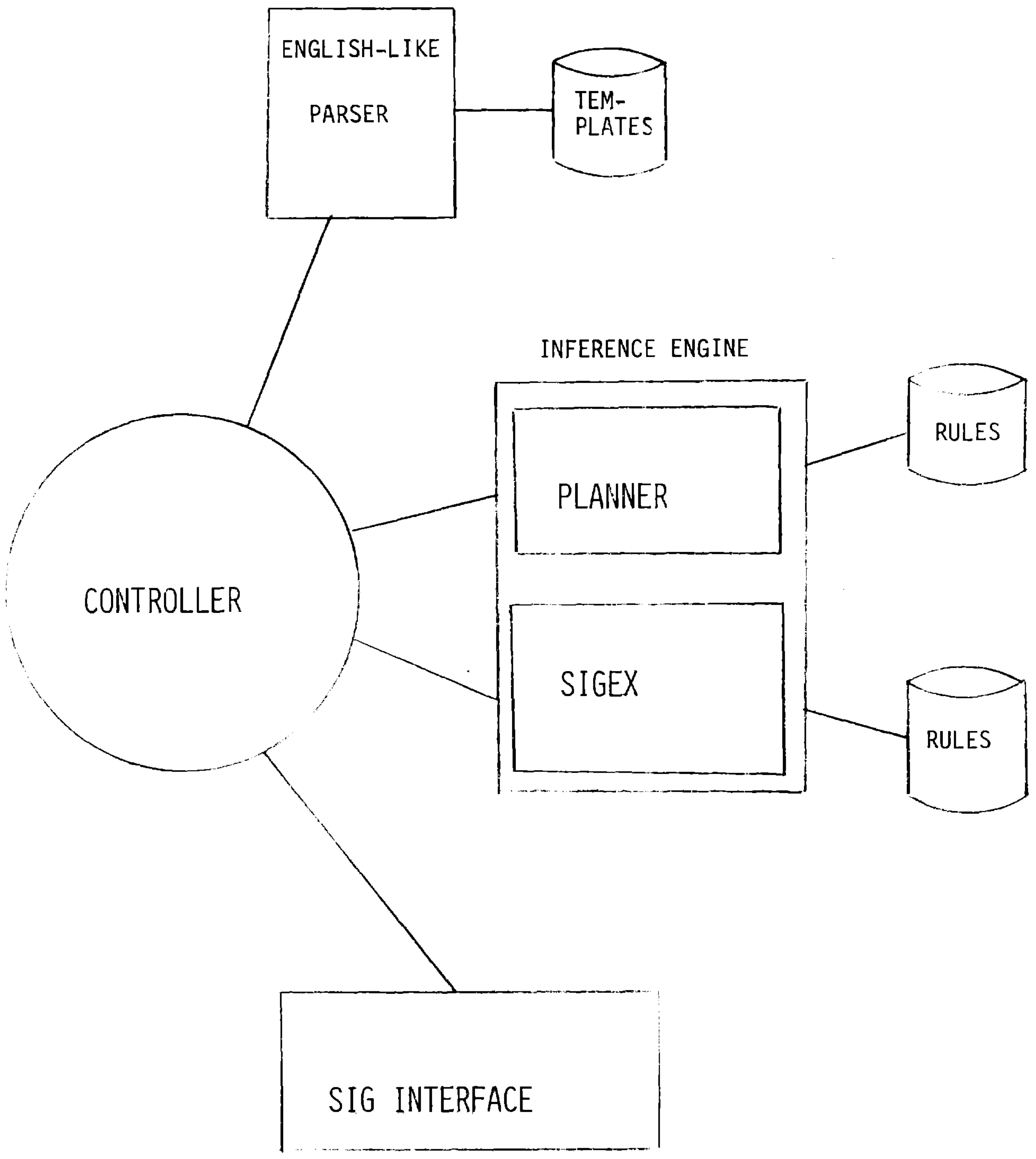
4.0 Implementation of SPEC

A conceptual model of SPEC is shown in figure 2. The system consists of a controller, an English-like parser, a SIG interface and two expert systems, PLANNER and SIGEX. The expert systems use a shared inference engine with their own set of rules.

4.1 Controller

The controller provides the shell for the modules of SPEC and controls the order in which events occur. When the system is started, the controller displays a prompt, reads the user input, and calls the parser which returns a parsed sentence and a goal. The controller then calls PLANNER which develops a suitable plan or returns an error message. If a plan is returned to the controller, the plan is passed to SIGEX one step at a time. SIGEX gathers any information needed to formulate a valid SIG command. Finally, the controller passes the SIGEX results to the SIG interface which creates a properly formatted SIG command, writes the command in a file and passes the file to SIG.

Figure 2. Conceptual Model of SPEC processing modules



4.2 English-like Parser

The user controls the system by entering sentences or sentence fragments at the keyboard. SPEC gives the appearance of understanding the input if it is properly phrased, but, in reality, the input is simply passed to a parser which looks for certain words which the system recognizes as keywords. The keywords are used as the goals for PLANNER. The user can include any information he wishes and the information will be used to set parameters which are used later. If user does not include enough information, either PLANNER or SIGEX will solicit the information.

When the system is started the English-like template file, which contains keywords and associated phrases, is read into the system. A typical template is shown in figure 3, and the format of the template file is shown in Appendix B. When a sentence is entered by the user, the sentence is scanned for a keyword. If such a keyword is found, then the sentence is further matched against phrases which are associated with that keyword.

The use of phrases allows the system to handle domain dependent words and phrases easily and to allow synonyms. These phrases serve as templates so that expected parameters can be assigned values from the user input. For example, if the user types the word "TIME" after the keyword PLOT has been typed, then the system will assign the parameter PLOT_TYPE the value TIME.

To further illustrate the operation of the parser, the template associated with the Fourier transform command is shown in figure 3. If the user wanted the system to do a Fourier transform on a file and plot the result, the user could type the following sentence:

PLEASE DO AN FFT ON FILE ABC AND FILTER IT <CR>

The parser first recognizes the word FFT and ignores the preceding words PLEASE, DO and AN as "noise" words. Then the parser looks for phrases which match those in the template file for the keyword FFT. The parser finds ON FILE which according to the template is equivalent to OF FILE or FILE or FILENAME and assumes that the word following ON FILE is the file name. So the word ABC is assigned to the parameter SRC_FILE which is the parameter which represents the name of the source file. Since the sentence contains a conjunction AND the process is started over. A second goal is defined, namely FILTER. However, since the file name is not repeated, the file name ABC is assumed to apply to keyword FILTER. The word IT in the above sentence is ignored as a "noise" word.

The intent of using such an English-like parser is to eliminate the need for the user to remember a complex syntax or to wander through a menu hierarchy. It is assumed that the user will be familiar enough with the language of the domain so that he can construct sentences which adequately express his intention. Several examples of the type of dialogue the system can handle are presented in Appendix A.

Figure 3. Typical English-like templates

```
( (H HELP = COMMAND HELP)     )


( (Q QUIT STOP DONE BYE  = COMMAND QUIT)    )


( (FFT (FOURIER TRANSFORM) = COMMAND FFT)
    ((ON FILE) (OF FILE) = SRC_FILE ?)
    (ON OF = QUAL ?)
    (FILE FILENAME = SRC_FILE ?)
    ((DATA_TYPE TIME) TIME = DATA_TYPE TIME)
    (FFT_SIZE (FFT SIZE) = FFT_SIZE ?)
    ((DATA_TYPE FREQ) FREQ  = DATA_TYPE FREQ)
    ((HEADER PRESENT)= HEADER PRESENT)
    ((HEADER ABSENT)= HEADER ABSENT)
    (HEADER = HEADER PRESENT)
    (FORMAT = FORMAT ?)
    (DELTA_TIME (DELTA TIME) DT = DELTA_TIME ?)
    (INIT_TIME (INIT TIME) TINIT = INIT_TIME ?)
    (DELTA_FREQ (DELTA FREQ) DF = DELTA_FREQ ?)
    (INIT_FREQ (INIT FREQ) FINIT = INIT_FREQ ?)    )
```

4.3 Inference Engine

The system contains two expert systems: PLANNER and SIGEX. Both of these systems use the same inference engine but work on different rules. The engine uses the rules in conjunction with data from the user and a backward chaining search technique to reach conclusions and execute custom LISP code routines.

4.3.1 Rules and Objects

When the system is loaded, the rules and attributes of the objects that are referenced in the rules are read from a disk file. The format of the rule file is shown in Appendix C. Basically, each rule consists of the following IF-THEN-EXECUTE format:

```
(RULE id IF (premise) ... (premise)
        THEN (conclusion) ... (conclusion)
        EXECUTE (action) ... (action) )
```

A premise or a conclusion has the following format:

```
     (object-type attribute value)
```

An object-type is a class of objects; for example, in SPEC one object-type is a source file, referred to as SRC_FILE. An object-type may have many occurrences of object-types and whichever object is last referenced is the current object for that object-type. For example, if the source file being processed is ABC then the current object of the object-type SRC_FILE is ABC.

Each object-type, and therefore its associated objects, is characterized by a set of attributes. For example, a SRC_FILE object has an attribute called DATA_TYPE which identifies the type of data in the file represented by the object. For each attribute of an object-type there may also be some associated information: a procedure, a question and a range. The procedure is a function which is executed by the inference engine when it can not find a value for that attribute. The procedure is written by the designer and must be written in LISP.

If there is no procedure specified, then the engine will attempt to solicit the value of an attribute from the user by printing a question. If the designer has included such a question with the attribute then that question is used; otherwise the system will try to formulate a question. Finally, if the user is asked for a value of an attribute, the system will verify that the value is one of an acceptable range of values if the designer has included such a range.

An action has the following format:

```
          (function-name any text)
```

Where function-name is the name of a function which will be given control after all of the attributes of the object-types in the conclusions are set to the indicated values. The entire expression will be passed to the function.

4.3.2 Operation of the Inference Engine

When SPEC is started it passes to the inference engine a goal which is a conclusion in at least one rule. The engine first checks to see if the the goal is already true. If so, the request is terminated.

If the goal is not already true, then the engine finds all the rules whose conclusions will make the goal true. Next, the engine selects the first rules in the list and tries to determine if the premises which make up the rule are true. There are two ways that a premise will be considered true. First, a premise will be true if the value of the attribute of the current object of the object-type referenced in the premise is equal to the value in the premise of the rule. Second, if the value in the premise statement is the special symbol "?", then the premise will be true if there is ANY value present for the attribute of the current object of the object-type referenced. If all the premises are true, then the attributes of the current object of the object-type named in the conclusions are set to the value specified in the conclusions. Finally, any actions specified in the rule are executed.

If the premises are not true, the rule is not true and other rules are tried. However, if the attribute of a premise is not known, then the backward chaining search is started. The system uses the premise as the new goal and attempts to determine if the new goal is true. If the system can not find any rule to conclude a goal, then it has two options. If there is a procedure assigned to attribute in the premise, the procedure is executed. If there is no procedure, then the user is asked for the value of the attribute in the premise. If there is a question associated with the attribute and object-type then the system uses the question as the prompt.

4.4 PLANNER

The purpose of the PLANNER is to convert a goal into a sequence of subgoals which will be passed to SIGEX. After the controller calls the parser on the English-like input it calls PLANNER for each goal recongized by the parser. PLANNER uses the inference engine described in section 4.3 and its own set of rules to reach the goal. Typical PLANNER rules for the FFT goal are shown in figure 4.

For example, if the user requests the system to perform an FFT on his data, the PLANNER is passed the goal (PLAN NL FFT). When the PLANNER looks for rules which will satisfy this goal, it finds two rules PR31 and PR32. If the system has already read the requested data file, then the PLANNER uses rule PR31 to setup the subgoal GOAL1 which is reformated by the controller into the following goal for SIGEX:

```
        (SIGEX GOAL FFT)
```

In this case, the plan is really simple; it consists of one subgoal which is the same as the initial request. In addition, PLANNER assigns the number of the current source file to the parameter SRC_DS which will be used later by the SIG interface.

Figure 4. Typical PLANNER rules

```
(RULE PR31 IF (SRC_FILE ANY_TS *)
           THEN (PLAN NL FFT)
                (PLAN GOAL1 (FFT (SRC_DS SRC_FILE CURRENT_TS)))
            EXECUTE)



(RULE PR32 IF (SRC_FILE DATA_TYPE TIME)
           THEN (PLAN NL FFT)
                (PLAN GOAL1 (TSREAD))
                (PLAN GOAL2 (FFT (SRC_DS SRC_FILE RAW_TS)))
           EXECUTE)
```

If, however, the system has NOT read the source data file, then the PLANNER uses rule PR32 to setup the subgoals GOAL1 and GOAL2 which are reformated by the controller into the following goals for SIGEX:

```
(SIGEX GOAL TSREAD)
(SIGEX GOAL FFT)
```

The plan consists of two subgoals: first, read the file; and second, formulate an FFT using the current file as the source file. In addition, PLANNER assigns the number of the current source file to the parameter SRC_DS.

4.5 SIGEX

The purpose of SIGEX is to acquire all the parameters which will be needed by the SIG interface to create an actual SIG command. Because the planning of the entire request is done by PLANNER, the sequence of subgoals generated by PLANNER is correct for the initial request. Therefore, the rules for SIGEX can be designed to focus on the details of SIG commands and NOT on how to get a sequence of commands to happen in the right order. Typical rules for SIGEX are shown in figure 5.

After the controller calls PLANNER it puts each subgoal returned by PLANNER in the proper format and passes the subgoal to SIGEX. When SIGEX receives a subgoal it tries to find all the rules which will satisfy the subgoal. The rules contain premises which force the inference engine to ask the user for specific data if the data is not already known.

For example, if PLANNER requests that SIGEX perform an FFT, SIGEX through its rules and finds that rule SR5 satisfies the subgoal. The rule causes the system to ask the user for the size of the FFT which is assigned to the parameter FFT_SIZE. The parameter FFT_SIZE is passed indirectly back to the controller so that it can be used later by the SIG interface.

4.6 SIG Interface

The purpose of the SIG interface is to assemble all the parameters needed for a SIG command and to actually send the command to SIG. The interface consists of a table which contains a list of all the parameters SIG needs for each SIG command and the order in which the parameters must be passed to SIG. It also contains a list of all parameters which must be reset as a result of successfully completing a SIG command.

Figure 5. Typical SIGEX rules

```
(RULE SR1 IF (SRC_FILE HEADER PRESENT)
             (SRC_FILE NAME ?)
          THEN (SIGEX GOAL TSREAD )
               (SIGEX SIG_COMMAND (TSREAD (NAME SRC_FILE NAME)))
          EXECUTE )


(RULE SR2 IF (SRC_FILE HEADER ABSENT)
             (SRC_FILE NAME ?)
             (SRC_FILE DELTA_TIME ?)
             (SRC_FILE INIT_TIME ?)
             (SRC_FILE FORMAT ?)
          THEN (SIGEX GOAL TSREAD)
               (SIGEX SIG_COMMAND (TSREADW (NAME SRC_FILE NAME)
                                   (DELTA_TIME SRC_FILE DELTA_TIME)
                                   (INIT_TIME SRC_FILE INIT_TIME)
                                   (FORMAT SRC_FILE FORMAT)))
          EXECUTE)


(RULE SR5 IF (SRC_FILE FFT_SIZE *)
          THEN (SIGEX GOAL FFT )
               (SIGEX SIG_COMMAND (FFT (FFT_SIZE SRC_FILE FFT_SIZE)))
          EXECUTE )
```

When the interface receives a request from SIGEX it uses the information stored by SIGEX and the table to create a syntactically correct command for SIG. The final command is then written to the SIG command file. The LISP interpreter is then suspended through the use of the VMS interface in NIL, and a VMS DCL command file is executed. The DCL command file starts SIG and causes SIG to execute commands from the SIG command file. When SIG finishes control is returned to VMS and the LISP interpreter is restarted. The interpreter then resumes where it was suspended, and the controller continues.

5.0 Conclusions and Observations

The development of SPEC went through several intermediate programs, and the progress of these programs was demonstrated to interested personnel. When SPEC was completed it was demonstrated to potential users. These users are primarily interested in simple frequency analysis and, in fact, much of their work is repetitive. It was my hope that the users would see the utility of SPEC and want to use it for more complex problems which they had not been able to address because of lack of time to learn new signal processing codes and techniques.

However, the users had no experience with interactive signal processing codes such as SIG. At that time, their signal processing was done by batch runs on a CDC 7600 using an analysis code written in the mid-70's. However, there was another project underway to combine their data acquisition and signal processing operations into a single computer system which included menu-driven signal processing software. It may not be surprising, therefore; that the users expressed little interest in SPEC and felt that the anticipated system would adequately handle their signal processing problems.

Although it was decided not to develop the prototype SPEC into a working system, the project provided several insights into expert system technology and man-machine interfaces which may be applicable to future work. First, a menu system is a better choice as an interface if the set of problems being solved is small. If there really are only a few functions to be done, then the menu system may very well capture those functions in an easy-to-use manner with a small number of menus. In fact, SIG has been designed to handle this class of problems very well.

To handle those users who want to solve a wider range of problems than could be easily enumerated on menus SIG provides an interface using a command language and allows processing of the language via command files. With this command language users can easily generate procedures which perform a variety of ad hoc signal processing techniques. But to really take advantage of this capability the user must learn the intricacies of SIG command formats and understand signal processing very well.

The intent of SPEC was to first eliminate the need to learn what SIG commands do and the command formats and then to provide a framework upon which enough signal processing knowledge could be added later to eventually assist users in the development of specialized signal processing techniques. However, the amount of knowledge captured in SPEC was too small to generate interesting signal processing techniques. It is certainly true that SPEC succeeded in freeing the user from understanding or even knowing any SIG commands or the exact formats of the SIG commands. But it was too difficult to appreciate the "expertise" required to generate these SIG commands because the range of problems that the knowledge base could solve was so simple that a simple menu system or minimum knowledge of SIG commands could have been used just as well.

Second, even though SPEC executes as compiled code it is too slow for an interactive environment. There were two reasons for the slow speed; first, the computer resources were not dedicated to SPEC. The code was running on a time-shared VAX so response was not consistent. Second, even at its best, the performance was too slow because the code was not optimized in any way. One of the major problems was that linear search was used to find applicable rules.

Third, the English-like interface was useful but not much easier to use than an artificial language with a rigid syntax. The use of keywords and many special constructions made the language almost as difficult to use as memorizing simple SIG commands. For the limited domain of interest a menu system would have been easier. An English-like interface could be very useful if the range of problems being solved is too large for a simple menu, but the interface must have a truly natural language capability; the interface must allow users to formulate requests without construction limitations and must provide spelling correction.

Fourth, the system asked the user too many questions. To avoid making incorrect assumptions, the system checked each decision by asking the user to verify the each assumption. This verification technique reduced the chance of doing the wrong operation, but it made the interface too time consuming. The user could not simply enter a request and walk away. The user had to actually be present through each stage of the request. Some of this problem could have been avoided if the interface to SIG had processed several SIG commands at one time instead of a single command.

The system must make more "intelligent" assumptions on its own. For example, the system might tailor the questions to the "history" of the user. After a user has answered these types of questions several times, perhaps the system could assume that the user is an experienced user and not prompt for verification. Alternatively, the system could be designed to use default values unless values are explicitedly provided in the English input and to modify dynamically the value for defaults. The assumptions could be displayed for the user at some later time.

6.0 Summary

SPEC was a successful demonstration of the use of expert systems technology to assist users in the use of a signal processing code. The system established that it is possible to free the user from having to know the commands and command formats which are needed to do signal processing in the frequency domain using the code SIG.

However, SPEC did not provide a better interface for SIG users than the menu system. The system was too slow and the signal processing knowledge was too limited. Users could have just as easily solved this class of problems using a simple menu system.

Appendix A. Sample SPEC Dialogue

A.1 Plot fft of some data

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
         LINE CONTINUATION IS &

?  DO FFT ON FILE ABC <CR>

I WILL ASSUME THE FILE ABC CONTAINS TIME DATA ? (OK? Y OR N) Y <CR>

I WILL ASSUME THE FILE ABC IS ALREADY IN
THE PROPER SIG FORMAT ? (OK? Y OR N) Y <CR>

       [SIG reads the file ABC]

I WILL USE ALL THE POINTS IN SIGNAL IN THE FFT ? (OK? Y OR N) Y <CR>

        [SIG does fft on the file ABC]

 I WILL LABEL THE PLOT TEST ? (OK? Y OR N) Y <CR>
```

TYPE <CR> TO CONTINUE <CR>

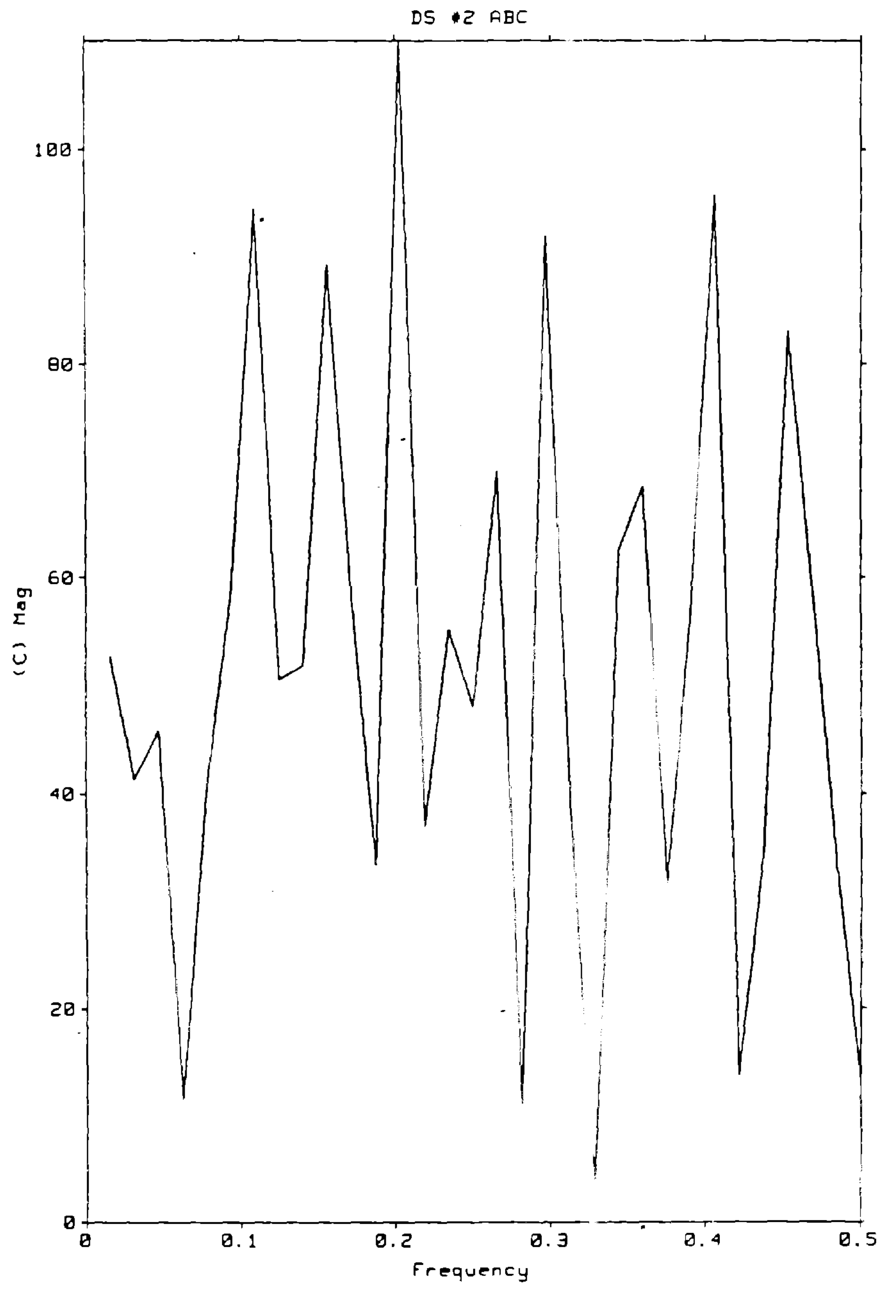


Figure A.1 FFT Plot of File ABC

A.2 Time plot

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
         LINE CONTINUATION IS &

 ?  LET ME SEE THE TIME PLOT <CR>

 I WILL LABEL THE PLOT TEST ? (OK? Y OR N) Y <CR>
```

```
TYPE <CR> TO CONTINUE <CR>
```

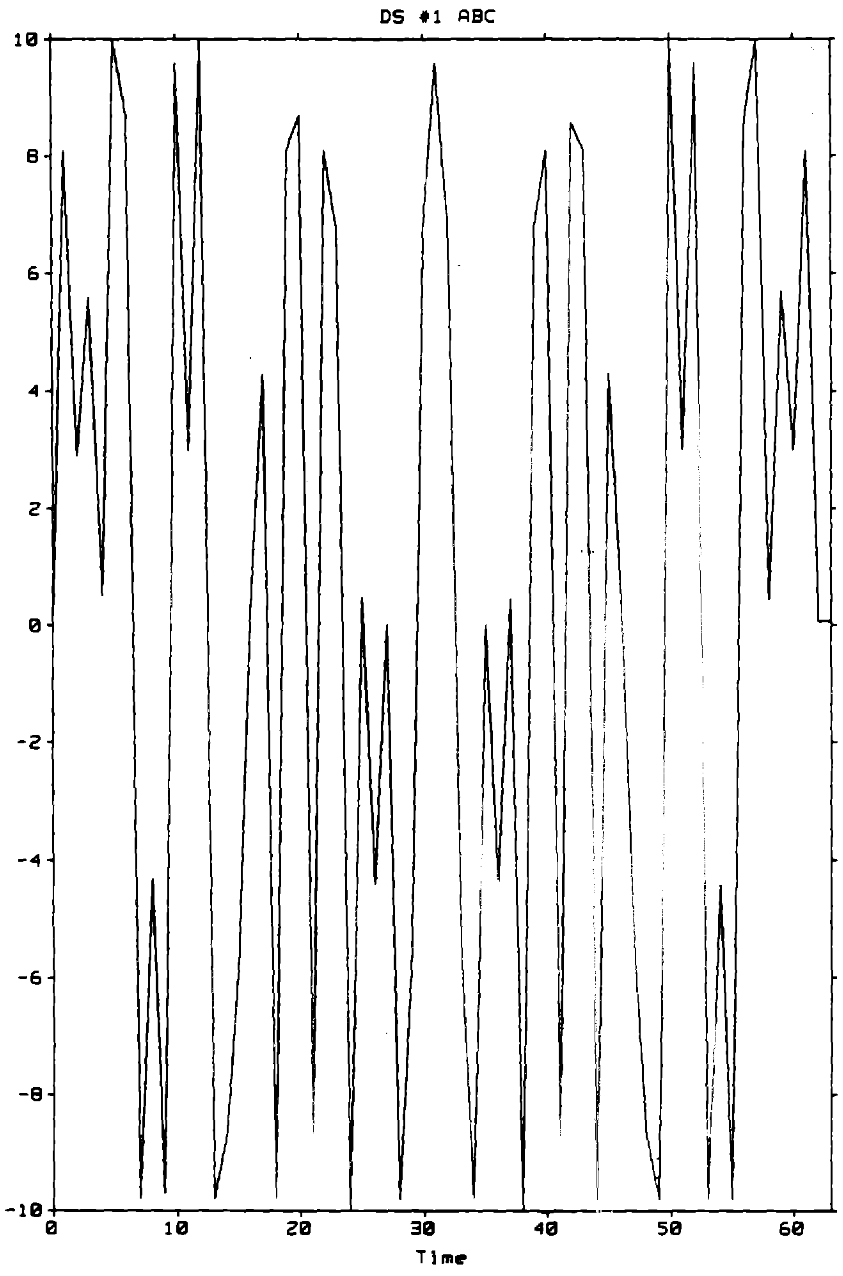


Figure A.2 Time plot of fi.e ABC

A.3 Plot another fft

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
         LINE CONTINUATION IS &

?  NOW DO THE FILE XYZ <CR>

I DON'T RECOGNIZE ANY WORD AS A COMMAND.
DO YOU MEAN TO DO THE COMMAND PLOT (Y OR N)?  N <CR>

WHAT DO YOU WANT TO DO?  FFT <CR>

I WILL ASSUME THE FILE XYZ CONTAINS TIME DATA ? (OK? Y OR N) Y <CR>

I WILL ASSUME THE FILE XYZ IS ALREADY IN
THE PROPER SIG FORMAT ? (OK? Y OR N) Y <CR>

       [SIG reads the file XYZ]

I WILL USE ALL THE POINTS IN SIGNAL IN THE FFT ? (OK? Y OR N) Y <CR>

       [SIG does fft on the file XYZ]

I WILL LABEL THE PLOT TEST ? (OK? Y OR N) Y <CR>
```

```
TYPE <CR> TO CONTINUE <CR>
```

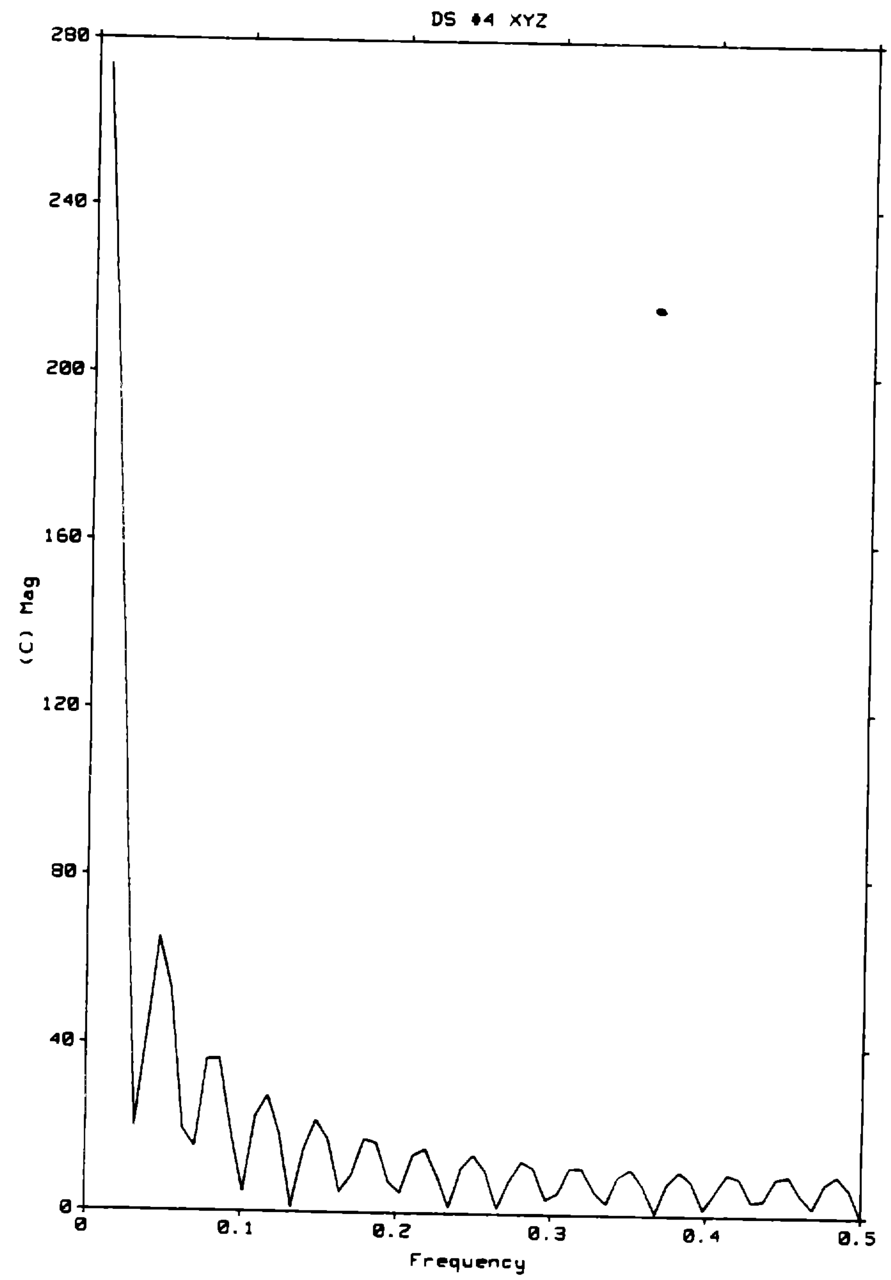


Figure A.3 FFT plot of file XYZ

A.4 Filter some data

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
         LINE CONTINUATION IS &

? FILTER FILE XYZ AND DO FFT AGAIN <CR>

I WILL USE A BESSEL FILTER ON THE FILE XYZ ? (OK? Y OR N)  N <CR>

WHAT TYPE OF FILTER DO YOU WANT TO USE ON FILE XYZ? HELP <CR>

VALID RESPONSES ARE AS FOLLOWS:
DEFAULT IS BESSEL
BESSEL BESS BES
BUTTERWORTH BUTTER BUT
HIT  <CR> TO CONTINUE <CR>

WHAT TYPE OF FILTER DO YOU WANT TO USE ON FILE XYZ? BESS <CR>

I ASSUME YOU WANT TO USE A LOW PASS FILTER ? (OK? Y OR N)  Y <CR>

AT WHAT FREQ DO YOU WANT TO THE FILTER TO BEGIN CHOPPING OFF THE DATA?
2<CR>

I WILL ASSUME A FIRST ORDER FILTER ? (OK? Y OR N)  Y <CR>

       [SIG does Bessel filter on file XYZ]

I WILL USE ALL THE POINTS IN SIGNAL IN THE FFT ? (OK? Y OR N) Y <CR>

        [SIG does fft on the filtered file XYZ]

 I WILL LABEL THE PLOT TEST ? (OK? Y OR N) Y <CR>
```

```
TYPE <CR> TO CONTINUE <CR>
```

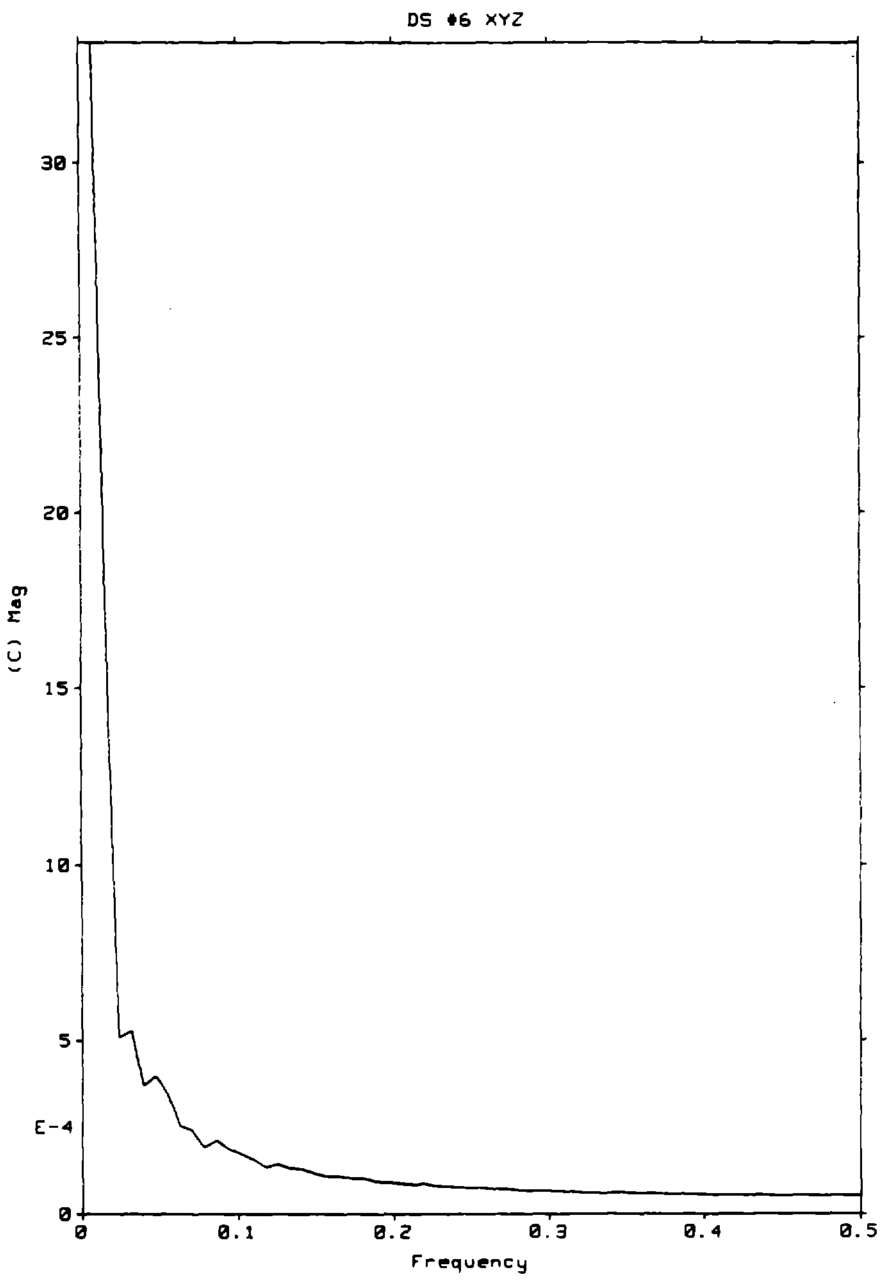


Figure A.4 FFT plot of filtered file XYZ

A.5 Plot original time data

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
         LINE CONTINUATION IS &

?  SHOW ME THE TIME PLOT OF ORIGINAL DATA <CR>

OK...

THE QUALIFICATION TYPE ORIGINAL IN YOUR REQUEST IS NOT
RECOGNIZABLE TYPE ?, OR QUIT, <CR> TO IGNORE IT, OR A DIFFERENT
QUALIFICATION  ? <CR>

 THE VALID QUALIFIERS ARE AS FOLLOWS:
    FITLER FILTERED
    FFT FOURIER_TRANSFORM
    IFF INVERSE_FFT
    PSD SPECTRUM
    RAW_TIME RAW_TS TIME
    RAW_FREQ RAW_FS FREQUENCY
  HIT <CR> TO CONTINUE <CR>

TYPE ?, OR QUIT, <CR> TO IGNORE IT, OR A DIFFERENT
QUALIFICATION <CR>

I WILL LABEL THE PLOT TEST ? (OK? Y OR N) Y <CR>
```

TYPE <CR> TO CONTINUE <CR>

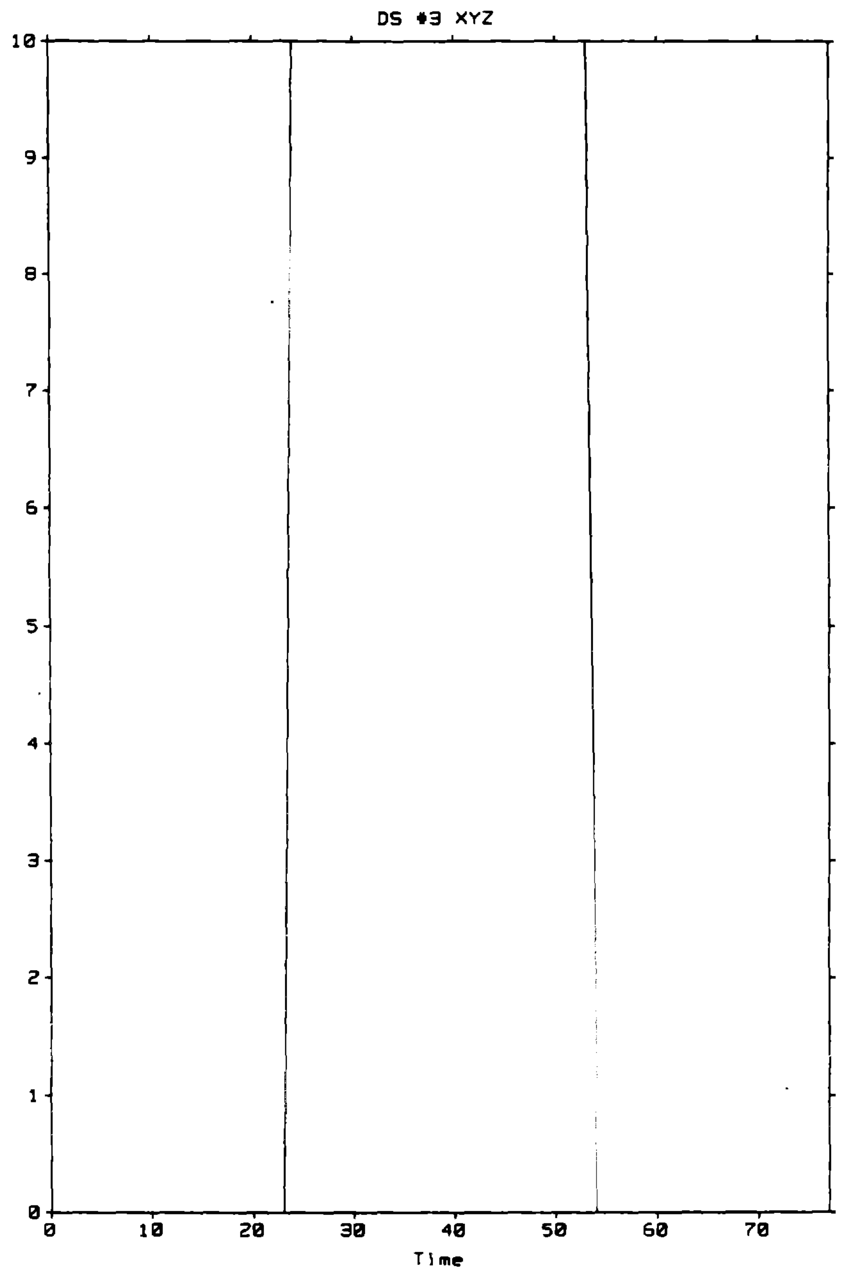


Figure A.5 Time plot of file XYZ

A.6 Exit SPEC

```
WELCOME TO THE SIGNAL PROCESSING EXPERT
         LINE CONTINUATION IS &


? QUIT <CR>


ALL DONE
NIL
```

Appendix B. Format of English-like Template File

```
( (keyword-statement) (phrase-statement) ... (phrase-statement) )

                            .
                            .
                            .

( (keyword-statement) (phrase-statement) ... (phrase-statement) )

((END))
```

WHERE keyword-statement has the following format:

```
     (keyword-list ... keyword-list = parameter value)

   WHERE keyword-list .... any literal or list of literals to be
                           used as a keyword
         parameter ....... any attribute to be passed to the
                           inference engine
         value ........... a literal to be passed as the value
                           of the attribute to the inference engine
```

WHERE phrase-statement has the following format:

```
     (phrase-list ... phrase-list = parameter value)

   WHERE phrase-list .... any literal or list of literals to
                          be used as a phrase
         parameter .......any attribute to be passed to the
                          inference engine
         value . . . . . .a literal to be passed as a
                          value the attribute to the
                          engine.  If the value is the  symbol "?"
                          then the next word from the English
                          sentence is used as the value.
```

APPENDIX C. Rule File Format

```
(OBJECT  object-type attribute procedure message-question range)

                          .
                          .
                          .

(OBJECT  object-type attribute procedure message-question range)



(RULE id   IF  (premise) ... (premise)
           THEN (conclusion) ... (conclusion)
           EXCUTE (action) ... (action) )

                          .
                          .
                          .

(RULE id   IF  (premise) ... (premise)
           THEN (conclusion) ... (conclusion)
           EXCUTE (action) ... (action) )



(END)
```

APPENDIX C. Rule File Format (continued)

```
WHERE object-type ....... the name of a class of objects
      attribute ......... a characteristic of the object
      procedure ......... the name of a function to be called to get
                          the value of the attribute
      message-question .. a prompt to get the value of the attribute
      range ............. list of possible values for the attribute

WHERE message-question has the following format:

                ((message) (question) )

      WHERE message ..... text to be printed before the question
            question .... text to prompt the user for the value of an
                          attribute

WHERE range has the following format:
   ( (DEFAULT value) (% predicate) (value (synonym ... synonym)) ... )

      WHERE DEFAULT ...... defines the value to be used if user wants
                           the default value for the attribute
            predicate .... the predicate which will be used to
                           value entered by the user.

WHERE id .......... unique identification for a rule
      premise ..... statement to check the value of an attribute
      conclusion .. statement which sets the value of an attribute
      action ...... function to be executed when the rule premises are
                    true

WHERE premise has the following format:

          (object-type attribute value)

      WHERE object-type ........ somé object-type
            attribute .......... characteristic of an object
            value .............. the literal value to be checked
                                 against the current value of the
                                 attribute for the object

WHERE conclusion has the following format:

          (object-type attribute value)

      WHERE object-type ........ some object-type
            attribute .......... characteristic of an object
            value .............. the literal value to be used to set
                                 the current value of the attribute
                                 for the object
```

```
WHERE object-type ....... the name of a class of objects
      attribute ......... a characteristic of the object
      procedure ......... the name of a function to be called to get
                          the value of the attribute
      message-question .. a prompt to get the value of the attribute
      range ............. list of possible values for the attribute

WHERE message-question has the following format:
                 (blank)
                 <line>
                 ((message) (question) )

      WHERE message ..... text to be printed before the question
            question .... text to prompt the user for the value of an
                          attribute

WHERE range has the following format:
   ( (DEFAULT value) (% predicate) (value (synonym ... synonym)) ... )

      WHERE DEFAULT ...... defines the value to be used if user wants
                           the default value for the attribute
            predicate .... the predicate which will be used to
                           value entered by the user.

WHERE id .......... unique identification for a rule
      premise ..... statement to check the value of an attribute
      conclusion .. statement which sets the value of an attribute
      action ...... function to be executed when the rule premises are
                    true

WHERE premise has the following format:

          (object-type attribute value)

      WHERE object-type ........ some object-type
            attribute .......... characteristic of an object
            value .............. the literal value to be checked
                                 against the current value of the
                                 attribute for the object

WHERE conclusion has the following format:

          (object-type attribute value)

      WHERE object-type ........ some object-type
            attribute .......... characteristic of an object
            value .............. the literal value to be used to set
                                 the current value of the attribute
                                 for the object
```

WHERE action has the following format:

```
        (function-name any text)

    WHERE function-name  ..... the name of a function to be
                               executed when all the premises of a
                               rule are true.
          any text ........... text to be passed to function
                               function-name
```

ACKNOWLEDGEMENTS

The author wishes to acknowledge the assistance of Terry Coupe on this project.